PSO 2

Recurrences and Trees

Announcements

TA Office hours has started, see ed

HW 1 due Thursday 11:59PM

From last week..

$$n^{\log n} \in \Omega(n!)$$

This was false. One way to see this:

$$n! = n \times (n-1) \times \ldots \times (n/2 + 1) \times (n/2) \times \ldots \times 2 \times 1$$

From last week..

$$n^{\log n} \in \Omega(n!)$$

This was false. One way to see this: **group terms**

n! = n x (n - 1) x … x (n/2 + 1) x (n/2) x … x 2 x 1

n! = (n x 1) x …

From last week..

$$n^{\log n} \in \Omega(n!)$$

This was false. One way to see this: **group terms**

$$n! = n \times (n-1) \times \ldots \times (n/2 + 1) \times (n/2) \times \ldots \times 2 \times 1$$

$$n! = (n \times 1) \times ((n-1) \times 2)$$

From last week..

$$n^{\log n} \in \Omega(n!)$$

This was false. One way to see this: **group terms**

$$n! = n \times (n - 1) \times \ldots \times (n/2 + 1) \times (n/2) \times \ldots \times 2 \times 1$$

$$n! = (n \times 1) \times ((n - 1) \times 2) \times \ldots \times ((n/2 + 1) \times (n/2))$$

From last week..

$$n^{\log n} \in \Omega(n!)$$

This was false. One way to see this: **group terms**

$$n! = n \times (n - 1) \times \ldots \times (n/2 + 1) \times (n/2) \times \ldots \times 2 \times 1$$

$$n! = (n \times 1) \times ((n - 1) \times 2) \times \ldots \times ((n/2 + 1) \times (n/2))$$

From last week..

$$n^{\log n} \in \Omega(n!)$$

This was false. One way to see this: **group terms**

$$n! = n \times (n-1) \times \ldots \times (n/2+1) \times (n/2) \times \ldots \times 2 \times 1$$

$$n! = (n \times 1) \times ((n-1) \times 2) \times \ldots \times ((n/2+1) \times (n/2))$$

Observe: all terms are ≥ n (there are n/2 terms)

From last week..

$$n^{\log n} \in \Omega(n!)$$

This was false. One way to see this: **group terms**

$$n! = n \times (n-1) \times \ldots \times (n/2+1) \times (n/2) \times \ldots \times 2 \times 1$$

$$n! = (n \times 1) \times ((n-1) \times 2) \times \ldots \times ((n/2+1) \times (n/2))$$

$$n! \geq n^{n/2}$$

Observe: all terms are ≥ n (there are n/2 terms)

Question 1

(Recursion Tree) Find a recurrence relationship which describes the running time of the following algorithms. For simplicity we will measure running times by the number of addition operations (+).

```
function Rec1(n : ℤ⁺)
    if n ≤ 0 then
        return n + n
    end if
    val ← 0
    val ← val + Rec1(n − 1)
    val ← val + Rec1(n − 3)
    return val
end function
```

```
function Rec2(n : ℤ⁺)
    if n ≤ 0 then
        return 0
    end if
    val ← 0
    for i from 1 to n − 1 do
        val ← val + Rec2(i)
    end for
    return val
end function
```

```
function Rec3(n : ℤ⁺)
    if n ≤ 0 then
        return n + n
```

```
function REC1(n : ℤ⁺)
    if n ≤ 0 then
        return n + n
    end if
    val ← 0
    val ← val + REC1(n − 1)
    val ← val + REC1(n − 3)
    return val
end function
```

(Recursion Tree) Find a recurrence relationship which describes the running time of the following algorithms. For simplicity we will measure running times by the number of addition operations (+).

||

Find T(n) = “number of times + is called when we run REC1(n)”

Recursive functions have a **base case** and a **recursive case**

Base case: T(__) = ______

```
function REC1(n : ℤ⁺)
    if n ≤ 0 then
        return n + n
    end if
    val ← 0
    val ← val + REC1(n − 1)
    val ← val + REC1(n − 3)
    return val
end function
```

Base case: T(0) = 2

(Recursion Tree) Find a recurrence relationship which describes the running time of the following algorithms. For simplicity we will measure running times by the number of addition operations (+).

II

Find T(n) = “number of times + is called when we run REC1(n)”

Recursive case: first calculate non-recursive work..

How many (non-recursive) +’s?

```
function REC1(n : Z+)
    if n ≤ 0 then
        return n + n
    end if
    val ← 0
    val ← val + REC1(n − 1)
    val ← val + REC1(n − 3)
    return val
end function
```

Base case: $T(0) = 2$

(Recursion Tree) Find a recurrence relationship which describes the running time of the following algorithms. For simplicity we will measure running times by the number of addition operations (+).

=

Find T(n) = "number of times + is called when we run REC1(n)"

Recursive case: first calculate non-recursive work..

How many (non-recursive) +'s? **2**

then count recursive calls

Recursive calls?

```
function Rec1(n : Z+)
    if n ≤ 0 then
        return n + n
    end if
    val ← 0
    val ← val + Rec1(n − 1)
    val ← val + Rec1(n − 3)
    return val
end function
```

How many (non-recursive) +'s? **2**

Recursive calls? **Rec(n - 1), Rec(n -3)**

→ T(n) = 2 + T(n - 1) + T(n - 3)

```
function Rec1(n : Z+)
    if n ≤ 0 then
        return n + n
    end if
    val ← 0
    val ← val + Rec1(n − 1)
    val ← val + Rec1(n − 3)
    return val
end function
```

(Recursion Tree) Find a recurrence relationship which describes the running time of the following algorithms. For simplicity we will measure running times by the number of addition operations (+).

Final answer:

T(0) = 2

T(n) = 2 + T(n - 1) + T(n - 3)

(important: include both base case and recursive case!)

```
function Rec2(n : ℤ⁺)
    if n ≤ 0 then
        return 0
    end if
    val ← 0
    for i from 1 to n − 1 do
        val ← val + Rec2(i)
    end for
    return val
end function
```

Base case: T(__) = ______

Recursive case:

How many (non-recursive) +'s?

Recursive calls?

```
function REC3(n : ℤ⁺)
    if n ≤ 0 then
        return n + n
    end if
    val ← 0
    val ← val + REC3(⌊n/2⌋)
    val ← val + REC3(⌊n/3⌋)
    for i from 1 to n − 1 do
        val ← val + 1
    end for
    return val
end function
```

Base case: T(__) = ______

Recursive case:

How many (non-recursive) +'s?

Recursive calls?

(Recursion Tree) Give a big-O closed form for each of the following recurrences. (Assume that $T(x) = 1$ for any $x \leq 1$.)

(1) $T(n) = 2T(n/4) + \sqrt{n}$

(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$

Unroll/use iterations?

(Recursion Tree) Give a big-O closed form for each of the following recurrences. (Assume that $T(x) = 1$ for any $x \leq 1$.)

(1) $T(n) = 2T(n/4) + \sqrt{n}$

(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$

Warning: Solving this $T(n)$ using iterations is a bad idea!

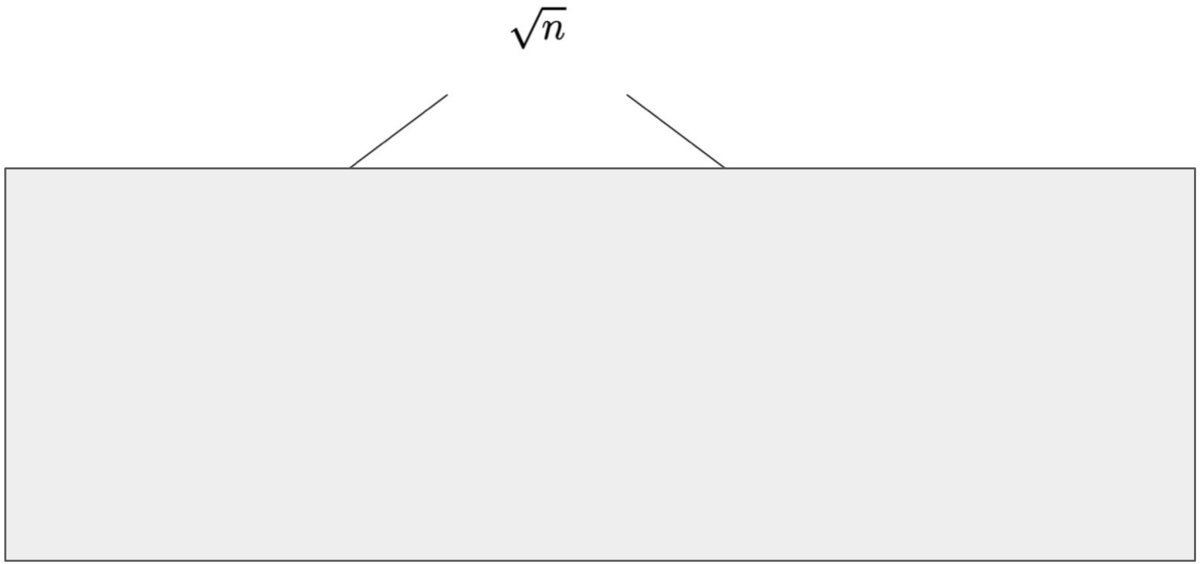
… kind of, we will see that trees help us organize better!

1. Draw out the tree
2. Find the cost at the ith level and the number of levels
3. Derive the sum and closed form

(Recursion Tree) Give a big-O closed form for each of the following recurrences. (Assume that $T(x) = 1$ for any $x \leq 1$.)

(1) $T(n) = 2T(n/4) + \sqrt{n}$

(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$



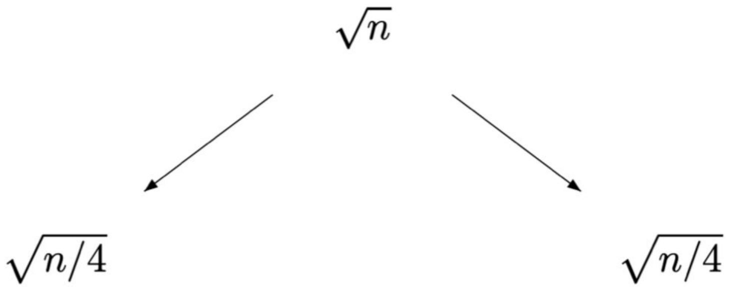
1. Draw out the tree
2. Find the cost at the ith level and the number of levels
3. Derive the sum and closed form

Question 1

(Recursion Tree) Give a big-O closed form for each of the following recurrences. (Assume that $T(x) = 1$ for any $x \leq 1$.)

(1) $T(n) = 2T(n/4) + \sqrt{n}$

(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$



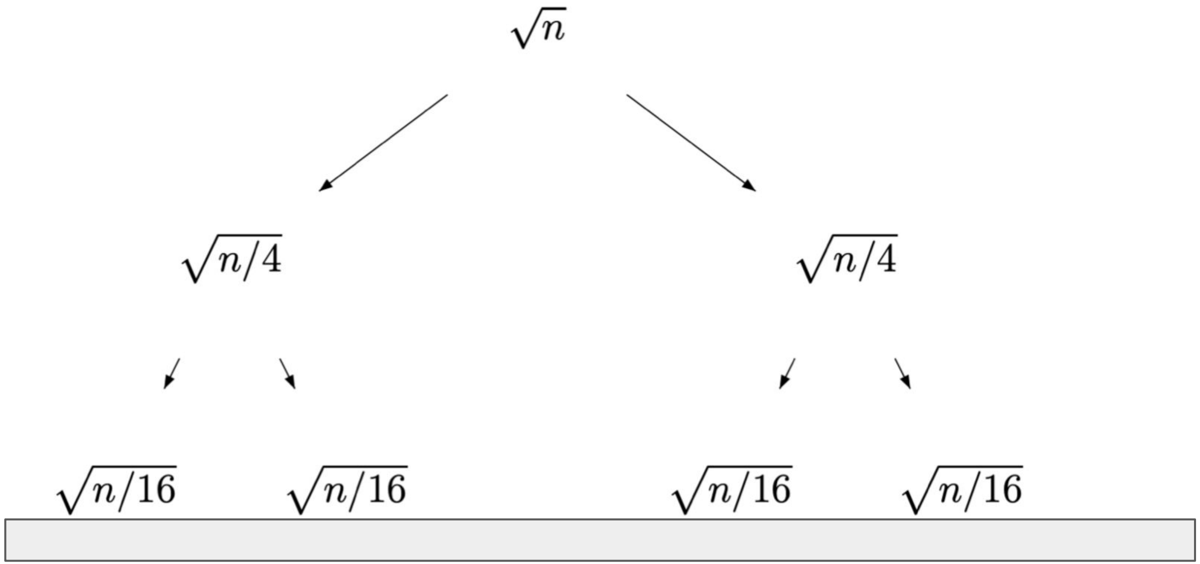
1. Draw out the tree
2. Find the cost at the ith level and the number of levels
3. Derive the sum and closed form

Question 1

(Recursion Tree) Give a big-O closed form for each of the following recurrences. (Assume that $T(x) = 1$ for any $x \leq 1$.)

(1) $T(n) = 2T(n/4) + \sqrt{n}$

(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$



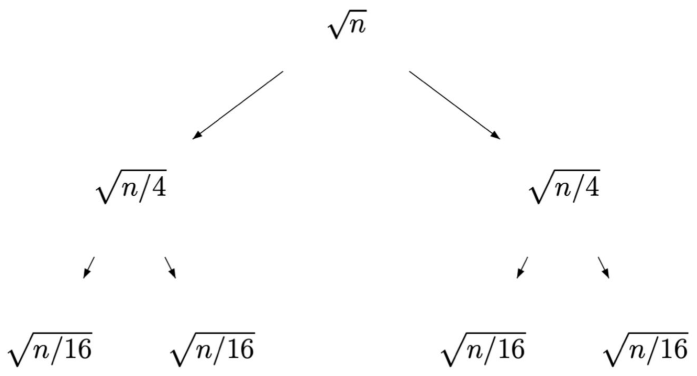
1. Draw out the tree
2. Find the cost at the ith level and the number of levels
3. Derive the sum and closed form

Question 1

(Recursion Tree) Give a big-O closed form for each of the following recurrences. (Assume that $T(x) = 1$ for any $x \leq 1$.)

(1) $T(n) = 2T(n/4) + \sqrt{n}$

(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$



1. Draw out the tree
2. Find the cost at the ith level and the number of levels
3. Derive the sum and closed form

Cost at first level:
Cost at second level:
Cost at ith level:

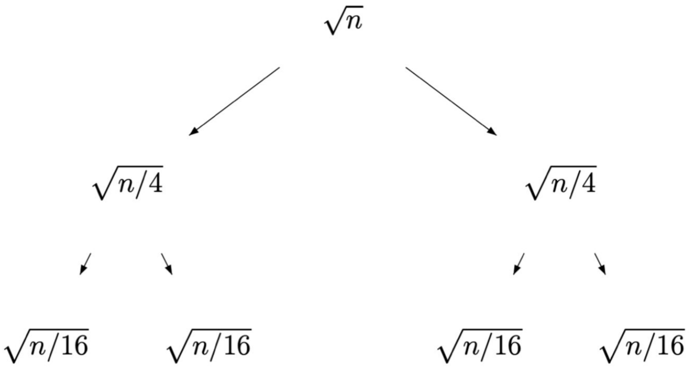
levels:

Question 1

(Recursion Tree) Give a big-O closed form for each of the following recurrences. (Assume that $T(x) = 1$ for any $x \leq 1$.)

(1) $T(n) = 2T(n/4) + \sqrt{n}$

(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$

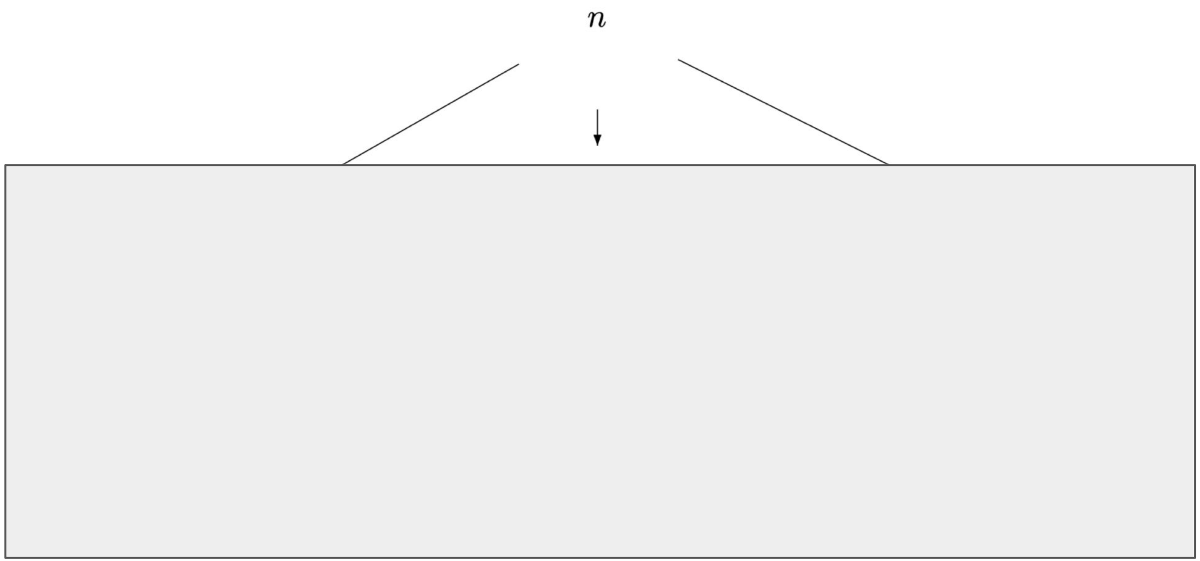


Cost at ith level: $\sqrt{n}$

Number of levels: $\log_4 n$

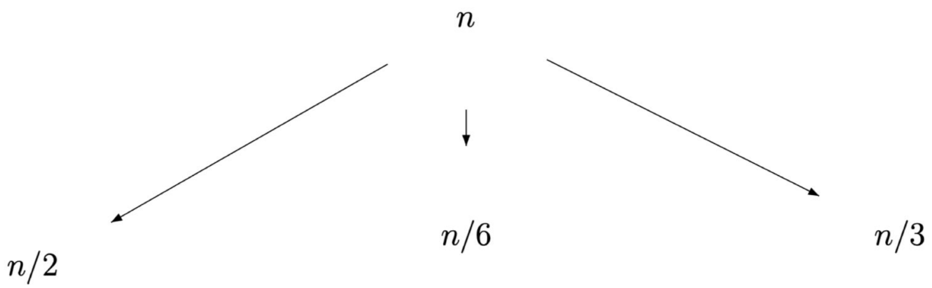
1. Draw out the tree
2. Find the cost at the ith level and the number of levels
3. Derive the sum and closed form

(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$



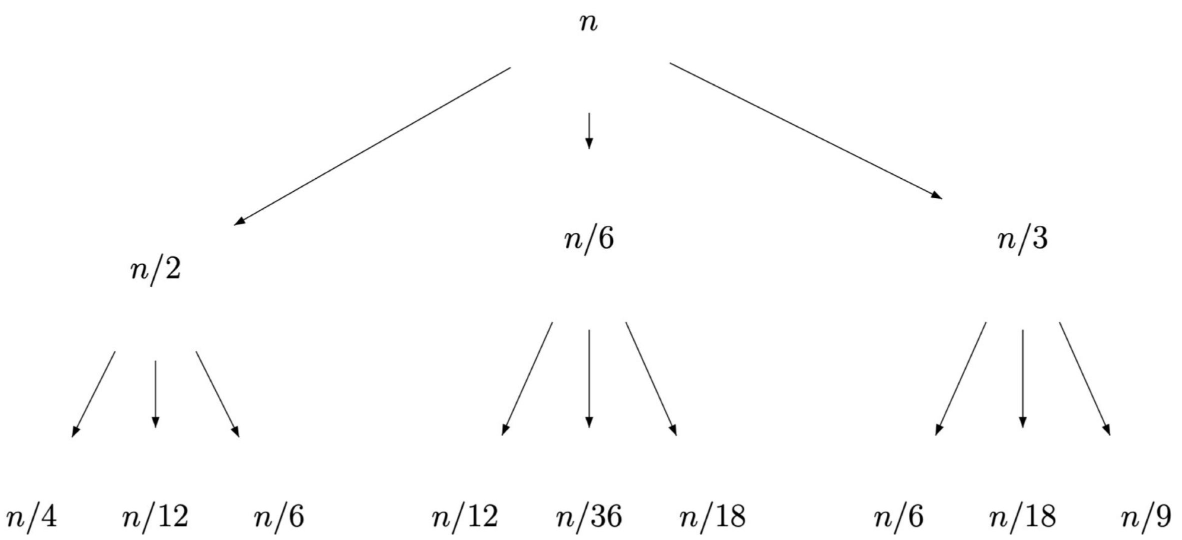
1. Draw out the tree
2. Find the cost at the ith level and the number of levels
3. Derive the sum and closed form

(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$



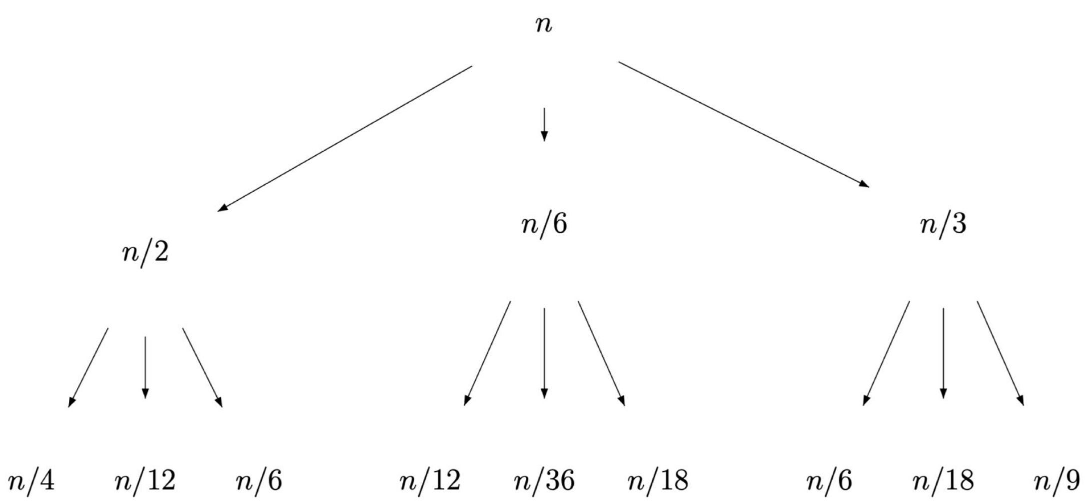
1. Draw out the tree
2. Find the cost at the ith level and the number of levels
3. Derive the sum and closed form

(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$



1. Draw out the tree
2. Find the cost at the ith level and the number of levels
3. Derive the sum and closed form

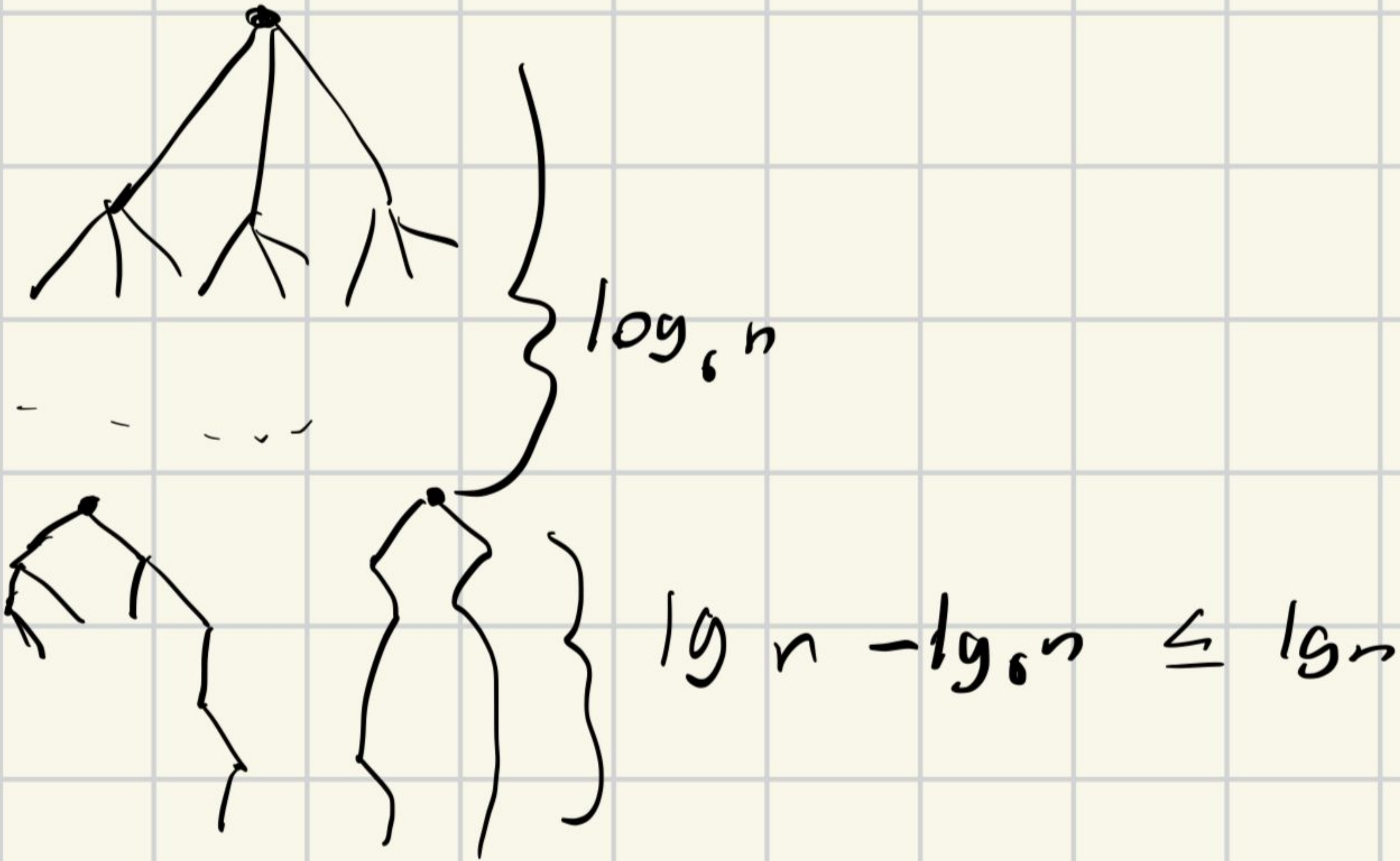
(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$



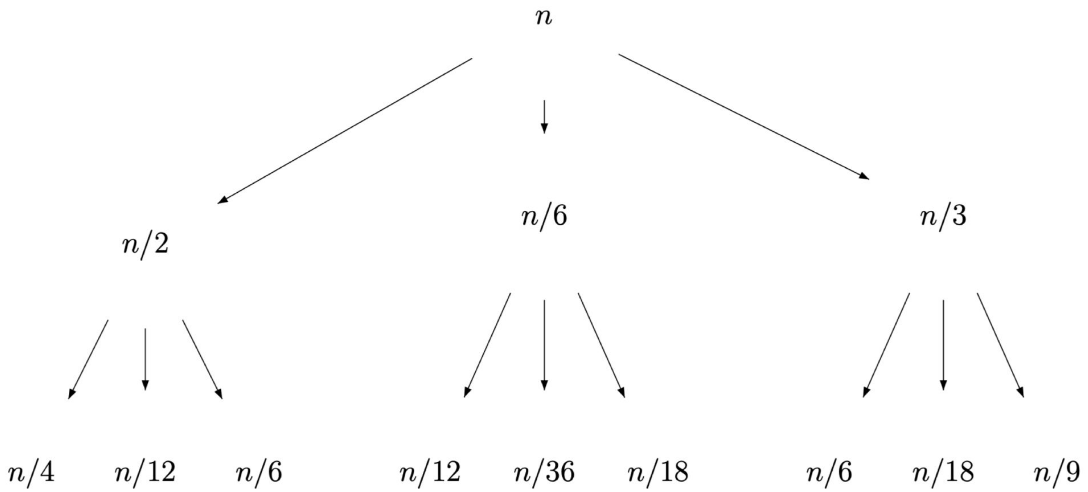
1. Draw out the tree
2. Find the cost at the ith level and the number of levels
3. Derive the sum and closed form

Cost at first level:
Cost at second level:
Cost at ith level:

levels:


log_6 n
lg n - lg_6 n ≤ lg n

(2) $T(n) = T(n/2) + T(n/3) + T(n/6) + n$



1. Draw out the tree
2. Find the cost at the ith level and the number of levels
3. Derive the sum and closed form

Cost at ith level: n

Number of levels: O(lgn)

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

What is the problem with a tree?

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

We usually like recurrences of this form

$$S(n) = \alpha S(n/\beta) + f(n),$$

E.g question 1 recurrences

Solution: Variable change! But to what value?

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

We usually like recurrences of this form $S(n) = \alpha S(n/\beta) + f(n),$

Change variable: m =

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

We usually like recurrences of this form

$$S(n) = \alpha S(n/\beta) + f(n),$$

Change variable: m = log n

$$T(2^m) = 2T(2^{m/2}) + m.$$

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

We usually like recurrences of this form

$$S(n) = \alpha S(n/\beta) + f(n),$$

Change variable: m = log n

$$T(2^m) = 2T(2^{m/2}) + m.$$

Change equation: S(m) =

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

We usually like recurrences of this form

$$S(n) = \alpha S(n/\beta) + f(n),$$

Change variable: m = log n

$$T(2^m) = 2T(2^{m/2}) + m.$$

Change equation: $S(m) = T(2^m)$

$$S(m) = 2S(m/2) + m,$$

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

We usually like recurrences of this form

$$S(n) = \alpha S(n/\beta) + f(n),$$

Change variable: m = log n

$$T(2^m) = 2T(2^{m/2}) + m.$$

Change equation: S(m) = T(2^m)

$$S(m) = 2S(m/2) + m,$$

This is just merge sort! O(mlogm) = O(log n * (log log n))

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

Intuition for O(log n * (log log n)) bound

First, how can we interpret T(n) = 2T(n / 2) + n?

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

Intuition for O(log n * (log log n)) bound

First, how can we interpret T(n) = 2T(n / 2) + n?

Read n in binary: $n_{\log n} \ldots n_2 n_1$

What is n / 2?

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

Intuition for O(log n * (log log n)) bound

First, how can we interpret T(n) = 2T(n / 2) + n?

Read n in binary: $n_{\log n} \ldots n_2 n_1$

What is n / 2?

Right shift: n / 2 = $n_{\log n} \ldots n_2 n_1$

I can only right shift log n times == log n tree height

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

Intuition for O(log n * (log log n)) bound

First, how can we interpret T(n) = 2T(n / 2) + n?

Right shift: n / 2 = $n_{\log n} \dots n_2 n_1$

I can only right shift log n times == log n tree height

Now, how does sqrt(n) look?

Read n in binary: $n_m \dots n_2 n_1$

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

Intuition for O(log n * (log log n)) bound

First, how can we interpret T(n) = 2T(n / 2) + n?

Right shift: n / 2 = $n_{\log n} \ldots n_2 n_1$

I can only right shift log n times == log n tree height

Now, how does sqrt(n) look?

Read n in binary: $n_m \ldots n_2 n_1$

Right shift ~(m/2) times : sqrt(n) = $n_m \ldots n_{m/2+1} n_{m/2} n_2 n_1$

I can only right shift log **m** times == log m tree height == loglogn tree height

Question 2

(Change a Variable) Give a big-O closed form for the following recurrence.

$$T(n) = 2T(\sqrt{n}) + \log n$$

Right shift ~(m/2) times : sqrt(n) = $n_m \ldots n_{m/2+1} n_{m/2} n_2 n_1$

I can only right shift log **m** times

== loglogn tree height

On each level, logn work, so T(n) = log(n) x loglog(n)